Introducing Access Control in Webdamlog

Serge Abiteboul, Émilien Antoine, Gerome Miklau, Julia Stoyanovich, Vera Zaychik Moffitt

▶ **To cite this version:**

Serge Abiteboul, Émilien Antoine, Gerome Miklau, Julia Stoyanovich, Vera Zaychik Moffitt. Introducing Access Control in Webdamlog. DBPL - 14th International Symposium on Database Programming Languages - 2013, Aug 2013, Riva del Garda, Trento, Italy. hal-00850754

HAL Id: hal-00850754
https://hal.inria.fr/hal-00850754

Submitted on 8 Aug 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Introducing Access Control in Webdamlog *

Serge Abiteboul
Émilien Antoine
INRIA Saclay & ENS Cachan
first.last@inria.fr

Gerome Miklau
INRIA Saclay & UMass Amherst
miklau@cs.umass.edu

Julia Stoyanovich
Vera Zaychik Moffitt
Drexel University
stoyanovich@drexel.edu
zaychik@drexel.edu

Abstract

We survey recent work on the specification of an access control mechanism in a collaborative environment. The work is presented in the context of the WebdamLog language, an extension of datalog to a distributed context. We discuss a fine-grained access control mechanism for intentional data based on provenance as well as a control mechanism for delegation, i.e., for deploying rules at remote peers.

1. Introduction

The personal *data* and favorite *applications* of a Web user are typically distributed across many heterogeneous devices and systems, e.g., residing on a smartphone, laptop, tablet, TV box, or managed by Facebook, Google, etc. Additional data and computational resources are also available to the user from relatives, friends, colleagues, possibly via social network systems. Web users are thus increasingly at risk of having private data leak and in general of losing control over their own information. In this paper, we consider a novel *collaborative access control mechanism* that provides users with the means to control access to their data by others and the functioning of applications they run.

Our focus is information management in environments where both data and programs are distributed. In such settings, there are four essential requirements for access control:

Data access Users would like to control who can read and modify their information.

Application control Users would like to control which applications can run on their behalf, and what information these applications can access.

Data dissemination Users would like to control how pieces of information are transferred from one participant to another, and how they are combined, with the owner of each piece keeping some control over it.

Declarativeness The specification of the exchange of data, applications, and of access control policies should be declarative. The goal is to enable anyone to specify access control.

To illustrate each of these requirements, let us consider the functionalities of a social network such as Facebook, in which users interact by exchanging data and applications. First, a user who wants to control who sees her information, can use a classic access control mechanism, such as the one currently employed by Facebook, based on groups of friends. Next, let us consider a user who installs an application. This typically involves opening much of her data to a server that is possibly managed by an unknown third party. Many Facebook users see this as unreasonable, and would like to control what the application can do on their behalf, and what information the application can access. Third, with respect to data dissemination, users would like to specify what other users can do with their data, e.g., whether their friends are allowed to show their pictures to their respective friends. Finally, the users want to specify access control on their data without having to write programs. Thus, this simple example already demonstrates the need for each one of the four above requirements.

From a formal point of view, we define an access control mechanism for WebdamLog, a declarative *datalog*-style language that emphasizes cooperation between autonomous peers [2]. We obtain a language that allows for declaratively specifying both data exchange and access control policies governing this exchange. There are different aspects to our access control:

- For extensional data, the mechanism is standard, based on access control lists at each peer, specifying who owns and who can read/write data in each relation of that peer.
- For intentional data, the mechanism is more sophisticated and fine-grained. It is based on *provenance*. In brief, only users with read access to all the tuples that participated in the derivation of a fact can read this fact.
- The previous two mechanisms are used by default, and we also support the means of overriding them.
- Finally, we introduce a mechanism for controlling the use of *delegation* in WebdamLog, which allows peers to delegate work to remote peers by installing rules, and is one of the main originalities of WebdamLog.

Note that access control is implemented natively as part of the WebdamLog framework. The main idea is to use extensional relations to specify access to extensional data. Thus accessibility of extensional facts is itself recorded as extensional facts, which can then be used by a WebdamLog program to *derive access* to intentional data.

Organization This short paper is organized as follows. The Webdamlog language is presented in Section 2. In Section 3, we present some aspects of access control. We conclude in Section 4.

* This work has been partially funded by the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC grantWebdam, agreement 226513. http://webdam.inria.fr/

2. Webdamlog

In [2], we introduced Webdamlog, a novel Datalog-style rule-based language. In Webdamlog, each piece of information belongs to a *principal*. We distinguish between two kinds of principals: *peer* and *virtual principal*. A peer, e.g., AlicePhone or Picasa, has storage and processing capabilities, and can receive and handle queries and update requests. A virtual principal, e.g., Alice or RockClimbingClub, represents a user or a group of users, and relies on peers for storage and processing. We further distinguish between *facts*, representing local tuples and messages between peers, and *rules*, which may be evaluated locally or delegated to other peers.

Webdamlog is primarily meant to be used in a distributed setting. Perhaps the main novelty of the language is the notion of *delegation*, which amounts to a peer installing a rule on another peer. In its simplest form, delegation is a remote materialized view. In its general form, it allows peers to exchange knowledge beyond simple facts, providing the means for a peer to delegate work to other peers. We will not describe Webdamlog in detail here, but will illustrate it with examples, referring the interested reader to [2].

The following are examples of Webdamlog facts:

```
agenda@AlicePhone(12/12/2012, 10 : 00, John, Orsay)
photos@Picasa(fileName : picture34.jpg,
              date : 09/12/2012, byteStream : 010001)
writeSecret@Picasa(login : Alice, password : HG − FT23)
```

The first fact represents a tuple in relation agenda on peer AlicePhone with information about an upcoming meeting, and the second, a photo in Alice's Picasa account (a tuple in relation photos on peer Picasa). The third fact represents Alice's login credentials for her Picasa account (in relation writeSecret on peer Picasa). Suppose that Alice wishes to retrieve, and store on her laptop, photos from Fontainbleau outings that were taken by other members of her rock climbing group. To this effect, Alice issues the following rule:

```
outingPhotos@AliceLaptop($pic) :-
  rockClimbingGroup@Facebook($member),
  findPhoto@AliceLaptop($member, $photos, $peer),
  $photos@$peer($pic, $meta),
  contains@$peer($meta, Fontainbleau)
```

This rule is a standard Webdamlog rule that illustrates various salient features of the language. First, the rule is declarative. Second, the assignment of values to peer names (e.g., $peer) and relation names (e.g., $photos) is determined during rule evaluation. Third, for $peer assigned to a system other than AliceLaptop (e.g., Picasa or Flickr), the activation of this rule will result in activating rules (by delegation), or in some processing simulating them in other systems. The evaluation of rules such as this one is performed by the Webdamlog system, which is responsible for handing communication and security protocols, and also includes a datalog evaluation engine, namely the Bud system [5].

The semantics of a Webdamlog rule depends on the location of the relations occurring in this rule. Let p be a particular peer. We say that a rule is *local* to p if the relations occurring in the body are all in p; intuitively, p can run such a rule. The effect of a rule will also depend on whether the relation in the head of the rule is local (to p) or not and whether it is extensional or intentional.

Generally speaking, Webdamlog supports the following kinds of rules.

- A. Local rule with local intentional head. These rules, like classical datalog rules, define local intentional relations, i.e., logical views.
- B. Local rule with local extensional head. These rules derive new facts that are inserted into the local database. Note that, by default, as in Dedalus [3], facts are not persistent. To have them persist, we use rules of the form m@p(U) :- m@p(U). Deletion can be captured by controlling the persistence of facts.
- C. Local rule with non-local extensional head. Facts derived by such rules are sent to other peers and stored in an extensional relation at that peer, implementing a form of messaging.
- D. Local rule with non-local intentional head. Such a rule defines a new intentional relation at a remote peer based on local relations of the defining peer.
- E. Non-local. Rules of this kind allow a peer to install a rule at a remote peer, which is itself defined in terms of relations of other remote peers. This is the *delegation* mechanism that enables the sharing of application logic by peers, for instance, obtaining logic (rules) from other sites, and deploying logic (rules) to other sites.

3. Access control in Webdamlog

We present three simple examples that highlight particular aspects of access control in WebdamlLog.

Fine-grained access control on intentional data Suppose that an intentional relation allPhotos@Alice has been specified by Alice. Suppose that Alice gives the right to friends, say Bob and Sue, to insert pictures into this relation. Alice's friends can do this by defining the rules:

[at Bob] allPhotos@Alice($f) :- bobPhotos@Bob($f)
[at Sue] allPhotos@Alice($f) :- suePhotos@Sue($f)

(Relation names bobPhotos and suePhotos are underlined to indicate that they are extensional.) allPhotos@Alice is intentional and is now defined as the union of bobPhotos@Bob and suePhotos@Sue.

The READ privilege on allPhotos@Alice is a prerequisite to having access to the contents of this relation, but access is also controlled by the provenance of each fact, making READ access fine-grained. One can think of each intentional fact as carrying its provenance, i.e., how it has been derived. In our simple example of Alice's album, the provenance of a photo coming from Bob will simply be the provenance token associated with the corresponding fact at Bob. Then, to be able to read a fact in allPhotos@Alice that is coming from bobPhotos@Bob, Charlie will need READ access on bobPhotos@Bob. To see a slightly more complicated example, suppose that a fact F may be obtained by taking the join of two base facts F_1, F_2; and that the same fact may be obtained alternatively by projection of a fact F_3. To access F a peer would need to have read access to its container (the relation that contains it) as well as to facts that suffice to derive it, here F_3 or the pair (F_1, F_2). In other words, a peer that has READ access to an intentional fact must have sufficient rights to derive that fact.

Overriding the default semantics For intentional data, we use by default an access control based on the full provenance of each fact. (If a fact is derived in several ways, each derivation specifies a sufficient access right.) Access control based on full provenance may be more restrictive than is

needed in some applications, and we provide the means to override it. Consider the following rule that Alice uses to publish her own photos to her friends:

```
[at Alice] allPhotos@$x($f) :-
        alicePhotos@Alice($f), [HIDE friends@Alice($x)]
```

Ignore the HIDE annotation first. This rule is copying the photos of Alice's friends into their respective allPhotos relations. A friend, say Pete, will be allowed to see one of Alice's photos only if he is entitled to read the relation friends@Alice. Now, it may be the case that Alice does not want to share this relation with Pete, and so Pete will not see her photos. The effect of the HIDE annotation is that the provenance of facts coming from friends@Alice is hidden. With this annotation, Pete will be able to see the photos. This feature is indispensable in preventing access control from becoming too restrictive.

Controlling delegation Recall that general delegation allows rules with non-local relations in the body. This leads to significant flexibility for application development and is the main distinguishing feature of the Webdamlog framework. It also creates challenges for access control.

The following example illustrates the danger of a simplistic semantics for non-local rules. Consider the two rules:

```
[at Bob] message@Sue("I hate you") :- date@Alice($d)
        aliceSecret@Bob($x) :-
            date@Alice($d), secret@Alice($x)
```

If we ignore access rights, by delegation, this results in running the following two rules at Alice's peer:

```
[at Alice] message@Sue("I hate you") :- date@Alice($d)
        aliceSecret@Bob($x) :-
            date@Alice($d), secret@Alice($x)
```

Assuming date@Alice($d) succeeds, then by the first rule Alice sends some hate mail to Sue, and by the second it sends the contents of the relation secret@Alice to Bob, even if Alice did not give READ access on this relation to Bob.

The main reason for this problem is that (by the standard semantics of Webdamlog) we are running the delegation rules as if they were run by Alice. Under access control, we are going to run them in a *sandbox* with Bob's privileges. So with the first rule, the hate message will be sent but marked as coming from Bob. And with the second, the data will be sent only if Bob has READ access to secret@Alice. So, for a client c delegating a rule to a server, the semantics of delegation under access control policies guarantees that:

- If the rule has side effects (e.g., it results in the insertion of tuples in the relation of another peer), the author of the update is c.
- The access privileges with which the rule executes are those of c.

Note that, in practice, Alice sends Sue a message saying that the author of the message is Bob. So, Sue may question this fact and asks Alice to prove that this is indeed the case. But if this is indeed the case, Alice has the delegation from Bob to prove her good faith.

Delegation is at the heart of distributed processing. With delegation, a peer p can ask another peer q to do some processing on its behalf. A natural question is whether this will yield exactly the same semantics (with possibly very different performance) as if p were getting the data locally and running a local computation. It turns out that the semantics is different. This is because q will use data that (i) q has access to; and (ii) p has access to (because of the sandboxing). On the other hand, a local computation at p is limited by (ii) but not by (i).

4. Conclusion

The WebdamLog language has been introduced in [2]. The system has been implemented and different aspects have been demonstrated in conferences [1, 4]. The access control mechanism is currently being implemented. The fine-grained mechanism for intentional data raises various issues. In particular, the materialization of intentional relations may generate lots of data if performed naively. This is the topic of on-going research.

References

[1] S. Abiteboul, E. Antoine, G. Miklau, J. Stoyanovich, and J. Testard. [Demo] rule-based application development using WebdamLog. In *SIGMOD*, 2013.

[2] S. Abiteboul, M. Bienvenu, A. Galland, and E. Antoine. A rule-based language for Web data management. In *PODS*, 2011.

[3] P. Alvaro, W. R. Marczak, N. Conway, J. M. Hellerstein, D. Maier, and R. C. Sears. Dedalus: Datalog in Time and Space. Technical Report UCB/EECS-2009-173, EECS Department, University of California, Berkeley, December 2009.

[4] E. Antoine, A. Galland, K. Lyngbaek, A. Marian, and N. Polyzotis. [Demo] Social Networking on top of the WebdamExchange System. In *ICDE*, 2011.

[5] J. M. Hellerstein. The declarative imperative: experiences and conjectures in distributed logic. *SIGMOD Record*, 39(1):5–19, 2010.